Date: 11-12-03 Express Mail Label No. EV 214917901 US

Inventors: Nicholas Stamos, Seth N. Birnbaum, Tomas Revesz, Jr., Donato Buccella, Keith A. MacDonald, Dwayne A. Carson and William Fletcher

Attorney's Docket No.: 3602.1000-002

MANAGED DISTRIBUTION OF DIGITAL ASSETS

RELATED APPLICATION

This application claims the benefit of U.S. Provisional Application No. 60/442,464 entitled "Method and System for Adaptive Identification and Protection of Proprietary Electronic Information," filed on January 23, 2003. The entire teachings of the above-referenced application are hereby incorporated by reference.

BACKGROUND OF THE INVENTION

Data security has been a significant issue facing system administrators since almost the inception of the data processing industry. Most computer users recognize the possibility of theft or misuse of data by unauthorized outsiders. The terms "hackers" and "crackers" are often used to describe outsiders who attempt to gain access to a system, and who are typically not involved in any way with an organization's operations, its internal employees or systems. Many different solutions already exist to protect an organization's data processing infrastructure from this kind of threat. These include physical access control, firewalls, sniffers and other network monitors, data encryption, intrusion detection systems and other solutions. These solutions are generally recognized as being adequate for their intended purpose most of the time.

However, there is a second class of computer users that also pose a security threat. Protection from these unauthorized insiders requires a different approach, but one that is also well known. Almost since the inception of disk-based storage systems, the concept of access control has been applied to limit the ability of certain users to access certain important files. Using these techniques, now a universal feature of almost every Operating System (OS), a desktop and/or network file server can provide for limited read, write, public, private and other types of access to files, directory structures and the like, depending upon permissions granted to particular users. Permissions can be attached to user accounts by a system administrator, based on the need to know, the departments in an organization of which a user is a member, and so forth.

Even when users obtain access to only a portion of a system, however, they can still use a variety of techniques to steal and/or damage information. These can include simple browsing for unsecured information in a network, and/or removal or deletion of information made available as a result of poor security practices. More sophisticated rogue insiders will employ network packet sniffers and/or spying software. A variety of approaches, such as centralized document and digital rights management systems, network auditing, and file management tools, can be effective tools against unauthorized use by insiders in some instances.

For example, U.S. Patent 6,510,513 issued to Danieli and assigned to Microsoft Corporation describes a security and policy enforcement system that utilizes a series of transactions between a server and a client using electronic security certificates. A first client generates a request for access to data by submitting a security certificate containing a digest to a trusted arbitrator server. The trusted arbitrator authenticates the first client's credentials and returns the security certificate. The data and security certificate are then combined to create a distribution, which, in turn, is acquired by a second client. The second client extracts the security certificate and generates a digest from the data in the distribution. If the digest from the second client matches the digest from the first client, then data is considered to be valid. Depending upon the certificate type and a policy level, the trusted arbitrator server can provide services such as notification of improper usage.

U.S. Patent 6,427,140 assigned to Intertrust Technologies is another type of digital rights management system. A system such as this was intended, however, to protect the rights of various participants in a transferring sensitive data, such as in an electronic commerce or other electronic facilitated transactions.

SUMMARY OF THE INVENTION

Neither of these solutions does much to protect misuse of information by authorized insiders. This class of users has a trusted status, as they are supposed to have access to important data files to carry out their assigned tasks. Thus, they are routinely granted permission to use such information on a daily basis. Their access to sensitive data therefore is not normally suspect. The problem comes when this class of trusted users abuse that trust - - by copying and then distributing sensitive information to outsiders or other unauthorized people. Such events can happen quite easily, and has been occurring with increasing frequency when a disgruntled or departing employee wishes to damage an organization.

What prior art security systems fails to account for is the fact that once granted access to sensitive information, it is quite easy for authorized users to distribute it in many different ways. The proliferation of Internet connections, e-mail, instant messaging, removable media storage devices, such as Compact Disk-Read Write (CD-RW) drives, Universal Serial Bus (USB) type memory and storage devices, and the like, make it a trivial task to copy vast amounts of information almost instantaneously. Other peripheral devices, such as wireless modems, wireless local network cards, portable computers, Personal Digital Assistants (PDAs), network tunnels, and the like, provide all too convenient vehicles by which an authorized user may distribute copies of files outside of the trusted system environment.

Even the most sophisticated file management systems cannot presently prevent such abuse. The root of the problem stems from the fact that once an authorized user opens a file, its contents are no longer controllable. Specifically, copies of the file contents may be taken "out of" the controlled environment of a network or file management system all too easily.

The present invention is intended to address security problems that originate when authorized users abuse their authority. The invention accomplishes this by restricting access at the point of use of a file, rather than simply controlling access at a storage point.

Specifically, an autonomous, independent agent process, such as running in the background of a client Operating System (OS) kernel, interrupts requests for access to resources. Such resource access requests may include, for example, requests to read a file, open a network connection, mount a removable media device, and the like. Since access is detected at the point of use, in the OS kernel interrupt mechanism, tracking of resource utilization will occur regardless of whether the original directly access request originated from an application program that is being executed by an end user, indirectly by applications on behalf of users, or even by system requests made independently of application software.

The autonomous independent agent process contains sensors that capture low level system events. These sensors may include operations such as file read, file write, file copy, clipboard cut, clipboard copy, CD-RW access, TCP/IP network message inbound, TCP/IP network message outbound and the like.

Low level events are then associated with one or more file names (handles) and filtered against an approved list. Thus, the raw events are filtered to remove references to files such as operating system files (.EXE, .DLL, etc. in the case of Microsoft Windows) and the like that do not contain sensitive application data. Only events relating to application files that may contain sensitive data are thus further tracked.

For example, an aggregate "FileEdit" event might be reported when a user has opened and modified a sensitive financial document, with that user then attempting to save it to a newly attached USB portable hard drive.

Other aggregate events can provide control over user access to files, applications, networks and peripheral bus interfaces. Such aggregate events are typically defined by a system administrator to actively enforce policies governing the acceptable use of system resources. Enforcement can then take place at the time and place of information access, via the agent process.

For example, the administrator may develop logic predicates that control the use of desktop applications, networks and interfaces that are otherwise unmanaged conduits for information leakage. These predicates may prevent, restrict, or require documentation for access to peripheral devices that are capable of storing massive amounts of information, such as Universal Serial Bus (USB), hard disk drives, Compact Disk-Read Write (CD-RW) burners, Instant Messaging (IM) applications and peer-to-peer network connections. Policies governing the use of these resources can be as detailed as required by the organization. Thus, the agent process' response to policy violations can be aggressive as the assessed risk demands, in accordance with the particular organization's requirements. The autonomous disconnected operation of the access control allows enforcements of policies to be carried out in a way that is as mobile as the user's other system.

Specific enforcement options can include: allowing the user action with a prompt for required user input (such as specifying a business justification); blocking the user action, or generation of a warning and/or an alert to a compliance or security manager. Optional warnings to a user and/or prompts for input reinforce the user's understanding and acceptance of acceptable use policies.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

Fig. 1 is a diagram illustrating traditional security perimeters in a data processing system and a point of use perimeter that can be implemented with the present invention.

Fig. 2 is a diagram illustrating how a Digital Guardian^{tm} server may provide policy predicates. (Digital Guardian^{tm} is a trademark of VerdaSys, Inc., of Waltham, Massachusetts.)

Fig. 3 is a process flow diagram illustrating the invention more particularly.

Fig. 4 is a table of possible low level atomic events.

Figs. 5A- 5C are a table of higher level aggregate events.

Fig. 6 illustrates point of use policy enforcement.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Fig. 1 is a diagram of a typical computer network 100 which consists of client devices 102 and servers 104 connected via local area network and/or inter-networking equipment. Connections to an outside network, such as the Internet 108, are made through devices such as routers or gateways 106. Connections through the Internet 108 can be also made to external computers 110.

Traditional security models attempt to prevent access by an untrusted outsider 110 to devices 102 and/or file servers 104 within the protected network 100. Thus, a network perimeter 120 is typically associated with network points of access, such as through router 106 and specifically at a firewall 107. The firewall 107 can thus prevent attempts by unauthorized users of outside computers 110 to access information stored in the server 104 or otherwise manipulate the local computers 102. Firewalls 107 can also establish a perimeter 120 for outgoing access such as, for example, by users attempting to access certain undesirable outside computers 110 that contain restricted or harmful websites, game servers, and the like.

Rather than establishing a perimeter at external points of physical access to a network, the present invention establishes a perimeter of accountability for file usage at the point of use. The accountability model can not only track authorized users of the computer 102 accessing files stored on a local server 104, but more importantly also monitors attempts to access or move such files to peripherals that distribute or record information, or other possible abuse events.

Such possible abuse events may occur whenever a user accesses devices which are not visible to or controllable by a local file server 104 or firewall 107. These events may include writing files to uncontrolled media such as Compact Disk-Read Write (CD-RW) drives 204, Personal Digital Assistants (PDA) 206, Universal Serial Bus (USB) storage devices 208, wireless devices 212, digital video recorders 214, or even printing of files. Other suspect events can include running external Peer-to-Peer (P2P) applications 201, sending files via external e-mail applications 202, running Instant Messaging (IM) applications, uploading files to web sites via the Internet 108, and the like. Thus, the invention can provide enterprise-wide monitoring of all file, application and network use. As will be understood shortly, the heart of this approach consists of a high level contextual stream that characterizes user activity as it occurs at the point of use, such as the desktop 102, and then potentially controlling its occurrence according to defined policies.

Turning attention to Fig. 2, the process for controlling distribution of digital assets will now be described in more detail. A first system component, called an agent process 300, is interposed between an Operating System (OS) 301 and applications 308 as they run on clients 102 and/or servers 104 within the network 101. The agent process 300 has sensors or shims to detect and track file, printing, clipboard, and I/O device operations, such as file read or write operations. These sensors may include, but are not limited to, file system sensor 502 (including CD/DVD burn sensor 510), network sensor 504, print sensor 505, clipboard sensor 506, API spy 508 and process spy 509.

While the clients normally include desktops 102-1 which have a direct wired (or wireless) connection 109 to the local network 101, the agent 300 may also run on disconnected client computers such as laptops 102-2, making a report of events once a connection is eventually made to the network 100.

The agent 300 reports atomic events 350 to an activity journaling process typically running on an activity journaling server 104-2. The journaling server 104-2 (also referred to herein as the Digital Guardian^{tm}) processes atomic event data and coalesces it into what are called aggregate events 360. Aggregate events 360 are detected when a certain predetermined sequence of atomic events occurs. Each aggregate event 360 is thus composed of one or more atomic events 350 that conform to some predetermined pattern indicative of activity that should be monitored.

Specific types and/or sequences of atomic events 350 that lead to aggregate events 360 will be described in detail later. It should be appreciated here, however, that the particular events reported and their aggregation types depend upon the specific activities sought to be monitored.

In addition, predicates 370 that define enforcement policies are forwarded to the agent process 300. These may originate from configuration commands entered at management console 102-5 and be sent through the Digital Guardian server 104-2.

To protect the network completely, the agent process 300 would typically reside on all desktops 102 and file servers 104 associated with an enterprise's networks. The activity journaling server 104 and agent process 300 may communicate through secure, networking based applications such as the Microsoft ".NET" infrastructure or other secure networking systems. The management console 102-5 also permits access to the database stored in the Digital Guardian server 104-2, and is used specifically to provide risk compliance, forensic reporting, and similar reports 310 to administrative users of the system.

The journaling server 104-2 may typically run within a Windows 2000 Server environment having a secure .NET framework. The journaling server 104-2 also has access to a database, such as Microsoft SQL Server 2000 for example, to provide record storage and retrieval functions. It is to be understood, of course, that the processes described herein can be implemented on other types of operating systems, server platforms, database systems, and secure networking environments.

Fig. 3 is a more detailed view of the client agent 300. These elements particularly consist of one or more sensors 500, file filter 520, and event coalescing/ aggregation 530, event bus 580, one or more policy predicates 585, policy enforcement engine 590, and I/O Request Packet (IRP) filter 595. It should be further noted that the agent process 300 can also provide real time evaluation and potentially enforcement of rules.

The agent 300 preferably runs as a kernel process in a client Operating System (OS). For example, the agent 300 may run within the kernel of Microsoft Windows 2000 or Windows XP. Autonomous operation of the agent 300 provides for detection of atomic events 350 even when client 102 is disconnected from the network 100. Any such events are reported when the client 102 is reconnected and can communicate with the Digital Guardian server 104-2.

In a preferred embodiment, the agent 300 will run multiple services under Windows so that if one service is stopped by a malicious user, the other one may restart the other process. The process is also hid from a task manager or similar processes in the operating system and will be able to work with safe mode boot features in order to guarantee full protection.

Atomic event sensors 500 report atomic events as an output when actions, typically associated with Input/Output (I/O) drivers, are intercepted at the OS kernel. The agent process 300 is therefore transparent to the end user and tamper resistant. The intercept may, for example, occur during an I/O Request Packet (IRP) in an interruptible kernel. Events may also be provided by Windows services and kernel level drivers.

The sensors 500 may include file operation sensor 502 (including CD/DVD burn sensor 510), network operation sensor 504, print queue sensor 505, clipboard sensor 506, Application Programming Interface (API) spy sensor 508 and other sensors such as process spy 509.

Data collected with an event depends on the event type, but can include:

- For invoked applications, the identity of the invoking process, executable name, start time, end time, and process owner
- For user operations, such as log on or log off, the time and user identification (ID)
- For file operations, source / destination file name, operation type (open, write, delete, rename, move to recycle bin), device type, first and last access time

- For network operations, source / destination address, port and host names, start/end time stamp, bytes sent and received, inbound and outbound data transmission times
- For CD-RW operations, file names, start/end times and amount of data transferred
- For printing operations, full path or file name, event start time or print job name
- For clipboard operations, destination process ID, event start time, full path of filename involved
- For other high level operations, such as access to removable storage media, file name, device ID, time of day, bytes transferred, and the like

An approved file filter 520 operates to automatically filter the dozens of inconsequential events generated by standard calls to system files. For example, it is quite common for many different .EXE and .DLL operating system files to be opened and accessed repeatedly in a typical executing Microsoft Windows application. In order to reduce the data flow to the journaling server 104-2, the file filter 520 uses an approved file list 522 to filter atomic (raw) sensor events 510.

The approved file list 522 may be implemented as a list of file names associated with events. However, in a preferred embodiment, the well known MD5 algorithm is used to generate a hash code for each file name. The MD5 hash code for a filename associated with an event is then matched against the approved list 522, rather than the complete file handle, to speed up the filtering process. Thus, only events associated with unapproved files are passed down to the coalescing stage 530.

The next stage is an atomic event coalescing stage 530 that attempts to aggregate atomic events 510. The coalescing stage 530 further filters atomic events 510 associated with or related to a single user action between the agent 300 and the Digital Guardian server 104. In general, applications frequently read small chunks of a file and not the entire file at the same time. For example, a user may open a 2 MegaByte (MB) spreadsheet file. However the OS may at a given time actually only access chunks of the spreadsheet file that are much smaller than that, such as 5 or 10 KiloBytes (KB) at a time. Thus, a typical pattern of access is to see a file open atomic event, followed by multiple read atomic events to the same file. If this sequence of atomic events is seen from the same process and the same executable with the same thread ID and the same file handle, event coalescing 530 will thus count only a single "FileOpen" event. In a preferred embodiment, there is a time attribute associated with event coalescing 530 such that if a time limit typically measuring in minutes of time is exceeded, at least one event will be reported between raw level events.

A comprehensive list of typical high level event patterns is shown in Fig. 4. For example, 43 different action types, some of which are low level atomic events and others which are high level aggregate events, are defined in the preferred embodiment. A given event is composed of several fields in the database, including perhaps an action type 571, level 572, event category 573, event name 574, event table ID 575, action detail 576, action detail value 577, and discriminants 578.

Event categories are associated with each event type. For example, in an event category “file”, event names include file read, file write, file rewrite, file copy, file rename, file delete, file move, file recycle, file restore. Similarly, network related events are TCP/IP inbound, TCP/IP outbound, USB inbound and so forth.

A scope is also associated with each event type. A scope is defined as either being a thread, process, login, machine, or all type scope. For example, “process” scope is an event that is consolidated into a high level event in the same process but not necessarily executing the same thread. “Machine” means that a reboot could occur between two events that occurred on the same machine.

Attributes commonly recorded for all high level events include an action type, an event count, bytes read count, bytes written count, event start, event end, and other possible actions. Source and destination hold numerous other attributes including the file, path, process, thread, and application identifying information that performed the event.

Other types of system events may include print events, disk write events, clipboard, user and machine events. The final type of low level event may be process events including process start and process end.

High level aggregate events are created by detecting a combination of the occurrence of low level events. More particularly, a high level aggregate event (action types 26-42) is determined after seeing a specific sequence of lower level events (action types 1-25). For example, action type 26 is a high level event called "FileEdited". This is an aggregate event that determines when a file has been edited. As the table indicates, the high level event aggregation process 570 may detect that a particular process, thread, and file has performed one or more reads to a particular file handle, followed by a write operation to the same process, thread and file handle. The event is then defined as an aggregate "File Edited" event.

Aggregate events are defined in greater detail in Figs. 5A, 5B and 5C. For example, a "Clipboard to File" aggregate event 510 is defined as detecting a clipboard cut or copy followed by a clipboard paste to file operation.

Similarly, a "BurnFile" event is associated with detecting a CD write atomic event followed by a file read atomic event. Thus, if a series of file reads are detected from one file handle, followed by a series of CD write events with the same process, the application is recognized as having written a file to a CD-RW.

Numerous other aggregate events are possible; the list in Figs 5A, 5B and 5C is only meant to illustrate a few of the many possibilities.

An event bus 580 serves to distribute if one or more aggregate events to predicate rules 585-1,..., 585-n. An event cache 582 may also share events occurring during a certain time interval for use by the predicates 585. Events are thus fed to one or more predicates 585, which serve to implement policy logic. As one example of a predicate, logic might be implemented to block peer-to-peer file transfer applications. A predicate 585-n can thus be developed to detect a "network open" event followed by a "read hard disk drive" event. As mentioned previously, the code to implement predicates 585 is typically downloaded from the Digital Guardian server 104-2, such as driving a user 100 boot process.

Another example of a multi-event control predicate 585 would be to prevent files that originate from a specific server from being removed from that machine. This involves placing the files on a tracking list to watch for all derivative file re-names, copies, etc, and remembering these derivative names as well as the where a user first identifies source when removal actions are attempted.

Another predicate 585 can be used for blocking of a file, or a series of files from being burned to CD/DVD. The process of burning files is a multi step process, to the file burning application of choice which file to burn, and then an actual attempt to burn them. So, a series of small file reads occur, followed by writing to a temporary file for buffering of data, followed by the actual burn operation.

When predicates 585-n detect a policy violation, notification is given to a policy enforcement engine 590. The policy enforcement engine 590 then takes steps to enforce the policies as represented by the predicate 585 at the point of use - - that is, within the user kernel. One such enforcement mechanism may be via kernel IRP control. Thus, events that result in an IRP request may be intercepted by the kernel and examined by the policy enforcement engine 590. If the requested IRP is not a violation, then the policy enforcement engine 590, through the IRP filer 595, allows the requested action to proceed.

However, if the requested action is a potential policy violation, then the policy engine 590 will take additional appropriate actions. These may include a number of actions 600, such as having the OS fail the IRP, such as action 601. The user and/or application thus interpret this as a failure of the specific operating system service or hardware device implicated in the request.

However, other types of actions can take place. For example, the operating system may generate a usual warning to the user that the requested action violates policy. This can occur by sending a message but then allowing the request to continue, as in a warn action 602.

A warn-requiring-reason action 603 is similar, but also requires the user to provide input to document the reason for the policy violation.

In a notify-block action 604, the user is notified that the request violated a policy and the action is blocked.

Finally, in a server alert action 605, an alert may be given to the journaling server 104-2 to indicate that the policy has been violated, providing additional information such as the identification of the user, the files involved and so forth.

It can now thus be seen how actions 600 which may occur as a result of the policy enforcement engine 590 may have a range of severities including simply failing the request, allowing the request to continue but logging the event, prompting the user for information as to why the policy is about to be violated and/or prompting with other restrictive and/or permission requesting notices.

Fig. 6 is an illustration showing how the policy enforcement engine 590 may implement a specific action 602, warn-requiring-reason 603, notify-block 604 and alert-action 605. For example, an enterprise policy may allow only designated types of printing to occur from a particular enterprise application 700. This policy is appropriately implemented with a notify-block action 604. The action 604 then blocks a requested print that falls outside of allowed policies.

Similarly, a block clipboard copy action 604-2 can be implemented to prevent clipboard copy operations from originating inside enterprise application 700.

A warn-on-file-copy action 602-1 has also been implemented which warns a user 102-1 when attempting to copy a file to a USB type device. The warn action 602-1 can send a message to the user but then allow the copy request to continue.

A policy may also be implemented to implement a warn-requiring-reason action 603-1 to prompt for a business purpose when a user attempts to burn a file to a CD. The action 603-1 causes a notice box 622 to be displayed at the user 102-1 requiring a response. Once the box is completed with a reason for the policy violation, the burn request is allowed to continue.

Finally, an alert action 605 may be implemented to control Instant Messaging (IM) file transfers. Here, information relating to the attempted activity can be provided to the journaling server 104-2 to indicate that the policy has been violated providing additional information, such as identification of the user, the files involved and so forth, in a way that is completely transparent to the user.

The policy enforcement engine 590 can also implement other policy schemes. For example, it may tally violations and take appropriate actions according to a "demerit" procedure. For example, one of the predicates 585 may be a demerit predicate that counts the number of times that a particular user attempts to violate a security policy. The demerit system can thus be used to determine when a user is attempting to "hoard" digital assets. For example, a user may be detected as opening or copying many files in a very short time period such as a day. If the source is a network share location and the user has for example copied greater than 100 MegaBytes (MB) in a single day, and counting predicate violations can rise to a more severe system report and denial of further usage.

With this system of actions, the applications and/or user do not necessarily know what is happening. That is, actions can be taken entirely with the operating system kernel which simply denies the IRP.

If the user confirms an action, this may also be used to provide evidence of the user's intent in a later analysis of a situation where valuable assets have been misappropriated.

While this invention has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims.